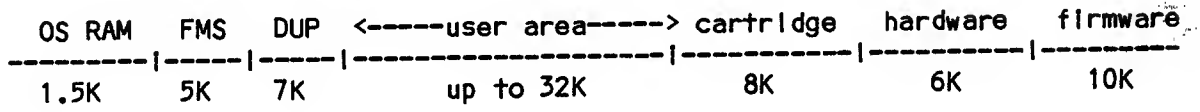
LECTURE NOTES

I. INTRODUCTION

A. Welcome
B. Names
C. Schedule
D. Handouts
E. Why develop software for the Atari?
   1. better machine; you can do more.
   2. ultimately a bigger market (consumers)
   3. better support

II. PCS SYSTEM OVERVIEW

A. A very people-oriented computer
B. Hardware
   1. 6502 (1.79 MHz)
   2. RAM (192K possible)
   3. ROM (10K OS, 8K or 16K cartridge)
   4. VIA
   5. POKEY sound, controller ports, serial bus
   6. ANTIC display microprocessor
   7. CTIA television output
C. Large memory map:



D. Strengths and weaknesses
   1. GRAPHICS!
   2. sound
   3. controllers
   4. 1.79 MHz 6502
   5. operating system (screen editing)
   6. BASIC variable names, indirection, syntax error handling
   7. BASIC strings
   8. slow disk
   9. little page zero or absolute RAM available
   10. maximum of five color registers in plain BASIC
   11. difficult to add hardware

III. ANTIC AND THE DISPLAY LIST

A. Background: how a TV works; definitions of terms

B. display system used by other personal computers
   1. microprocessor-->RAM-->hardware-->screen
   2. the 'hardware' part is the limiting factor
   3. PET, TRS-80: fixed RAM, single mode (text). simple hardware
   4. Apple: two RAM sizes, 3 fundamental modes, better hardware

C. The Atari display system
   1. functional differences
      a. 14 fundamental display modes
      b. modes changeable on screen
      c. screen RAM anywhere
   2. ANTIC, a video microprocessor
   3. ANTIC's instruction set
      a. display instructions (graphics, text, or blank)
      b. JMP and JVB
      c. special options (scroll, DLI, and LMS)
   4. the display list (ANTIC'S program)
      a. synchronized to television cycle
   5. screen memory
   6. ANTIC writes only to CTIA

D. A sample display list (all values in hex)

```
70   Skip 8 lines
70   skip 8 lines
70   skip 8 lines
42   'mode 0' line with Load Memory Scan
LO   address of screen memory
HI   address of screen memory
xx   here follow mode bytes
xx   must total 192 horizontal scan lines
xx
xx
xx
xx
xx
xx
xx
41   Jump and wait for vertical blank...
LO   ...to beginning of display list
HI   high byte of address of display list
```

E. Using display lists
   1. make a paper image
   2. mode line setup
   3. scan line count
   4. arranging screen memory
      a. mixed mode screen memory discontinuities
      b. maintaining integer multiples
      c. brute force: calculating addresses
      d. setting up independent windows

F. Uses of display list manipulations
   1. vertical spacing
   2. mixed modes (note problems above!)
   3. access to new modes
   4. screen sequencing
   5. dynamic display lists

IV. INDIRECTION: COLOR REGISTERS AND CHARACTER SETS

A. Indirection is harder to understand but more powerful

B. Color register indirection
   1. One number in a color register controls the color of many pixels
   2. less RAM consumption
   3. more colors to choose from
   4. color cycling (animation)
   5. logically keying colors to situations
   6. display list interrupt capability

C. Character set indirection
   1. Direct: character code gives character in ROM
   2. Indirect: character code gives character in any specified table
   3. Procedure
      a. define characters on graph paper
      b. encode into bytes
      c. stuff into RAM
      d. POKE CHBAS with address of charset
      e. must start on 1K boundary (1/2K for GR. 1 & 2 sets)
      f. 8 bytes per character
   4. Capabilities
      a. multiple character sets (fonts)
      b. graphics character sets; maps; mixed text and graphics
      c. 4 color character sets
      d. vertical reflect
      e. time changes of character sets
      f. human slow: change of scenery
      g. human fast: animation
      h. machine fast: DLI's
   5. Overall assessment

V. PLAYER-MISSILE GRAPHICS

A. The problem: Animation

B. The traditional solution: playfield animation
   1. move bytes through RAM to move screen image
   2. address calculations; very slow
   3. resultant limitations:
      a. pure horizontal motion
      b. pure vertical motion
      c. few objects moving
      d. slow motion
   4. Essence of problem: 2d image, 1d RAM

C. The Atari solution: player-missile graphics
   1. Fundamental idea: 1d image, 1d RAM
   2. map table of bytes directly from RAM onto screen
   3. image is a vertical column 8 bits wide
   4. vertical motion with 1d move routine
   5. horizontal motion by horizontal position register

D. Embellishments
   1. 4 players, each with its own color register
   2. Controllable player widths
   3. Two vertical resolutions possible
   4. missiles (two bits wide)
   5. image priorities

E. How to do it
   1. Set aside player-missile RAM area (1K or 2K)
   2. set player colors
   3. set player widths and positions
   4. draw in players
   5. enable through PMBASE and DMACTL
   6. see demo program HOBBY1

F. Capabilities
   1. Animated objects
   2. Additional color
   3. Special offline characters
   4. cursors
   5. additional resolution by priority control

VI. DISPLAY LIST INTERRUPTS

A. A very powerful capability

B. Fundamental ideas
   1. Screen drawing is time sequenced
   2. By cutting into draw at appropriate time, image can be changed
   3. change image by changing ANTIC registers
   4. Timing provided by the DLI

C. How a working DLI happens
   1. ANTIC encounters display list instruction with interrupt bit set
   2. ANTIC checks NMIEN for enabling bit
   3. If DLI is enabled, ANTIC interrupts 6502
   4. 6502 vectors through $0200, $0201 to your DLI service routine
   5. When done, 6502 resumes work.
   6. Process repeats at same point on screen each 1/60th second.

D. How to set up a DLI
   1. Write and place DLI service routine (page six?)
   2. Set DLI bit in appropriate display list instruction
   3. Set DLI vector in $0200, $0201
   4. Enable DLI with $C0 into $D40E
   5. Sample program: HOBBY2

E. Considerations
   1. colors into ANTIC, not shadow. Attract
   2. Execution time
   3. things happening halfway across line; (WSYNC); keyboard IRQ
   4. Multiple DLI's: separation problem, slow response
   5. Using VCOUNT

F. Capabilities
   1. Multiple playfield colors
   2. Multiple color, width, position, and priority players
   3. Multiple character sets; Rosetta Stone
   4. vertical screen architecture

VII. SCROLLING

A. Coarse scrolling
   1. method 1: move data through playfield with move routine.
   2. method 2: move playfield over data with LMS byte changes.
      a. serial scroll with single LMS instruction
      b. pure vertical scroll with single LMS instruction
      c. pure horizontal scroll with multiple LMS instructions
      d. mixing horizontal scroll with vertical scroll

B. Fine scrolling
   1. scrolls entire pixel line
   2. either horizontal or vertical or both
   3. line by line control
   4. DLI capability

C. How to do it
   1. set scroll bit in display list instruction
   2. store HSCROLL and VSCROLL values
   3. edge buffering
   4. coupling with coarse scrolling
   5. sample program: SCRL19.ASM

D. Applications
   1. Large images with screen window
      a. maps
      b. diagrams (schematics, blueprints)
      c. large blocks of text
      d. menus
      e. screen rotations (well, almost)

HSCROL		XX		XX		XX		XX		XX		XX		XX		XX	Horizontal Scrolling
VSCROL			XX	XX			XX	XX			XX	XX			XX	XX	Vertical Scrolling
LD MEM SCAN					XX	XX	XX	XX					XX	XX	XX	XX	Load memory scan (3 byte)
INST INTERRUPT									XX	XX	XX	XX	XX	XX	XX	XX	Display instruction interrupt
BLK 1	00								80								Blank 1 line
" 2	10								90								Blank 2 lines
" 3-7																	Blank 3 thru 7 lines
" 8	70								F0								Blank 8 lines
JMP	01								81								Jump (3 byte instruction)
JVB	41								C1								Jump & wait for Vert. Blank (also 3 byte)
CHR (40,2,8)	02	12	22	32	42	52	62	72	82	92	A2	B2	C2	D2	E2	F2	
" (40,2,10)	03	13	23	33	43	53	63	73	83	93	A3	B3	C3	D3	E3	F3	
" (40,4,8)	04	14	24	34	44	54	64	74	84	94	A4	B4	C4	D4	E4	F4	Character Mode Instructions
" (40,4,16)	05	15	25	35	45	55	65	75	85	95	A5	B5	C5	D5	E5	F5	
" (20,5,8)	06	16	26	36	46	56	66	76	86	96	A6	B6	C6	D6	E6	F6	
" (20,5,16)	07	17	27	37	47	57	67	77	87	97	A7	B7	C7	D7	E7	F7	
MAP (40,4,8)	08	18	28	38	48	58	68	78	88	98	A8	B8	C8	D8	E8	F8	
" (80,2,4)	09	19	29	39	49	59	69	79	89	99	A9	B9	C9	D9	E9	F9	
" (80,4,4)	0A	1A	2A	3A	4A	5A	6A	7A	8A	9A	AA	BA	CA	DA	EA	FA	
" (160,2,2)	0B	1B	2B	3B	4B	5B	6B	7B	8B	9B	AB	BB	CB	DB	EB	FB	Memory Map Mode Instructions
" (160,2,1)	0C	1C	2C	3C	4C	5C	6C	7C	8C	9C	AC	BC	CC	DC	EC	FC	
" (160,4,2)	0D	1D	2D	3D	4D	5D	6D	7D	8D	9D	AD	BD	CD	DD	ED	FD	
" (160,4,1)	0E	1E	2E	3E	4E	5E	6E	7E	8E	9E	AE	BE	CE	DE	EE	FE	
" (320,2,1)	0F	1F	2F	3F	4F	5F	6F	7F	8F	9F	AF	BF	CF	DF	EF	FF	

Number of TV lines per cell
Number of Colors (Background + Playfield types)
Number of Horizontal cells (standard width screen)

Figure II.3 DISPLAY INSTRUCTION OPCODES

BASIC OUTLINE

I. PRIMARY CODING GOALS

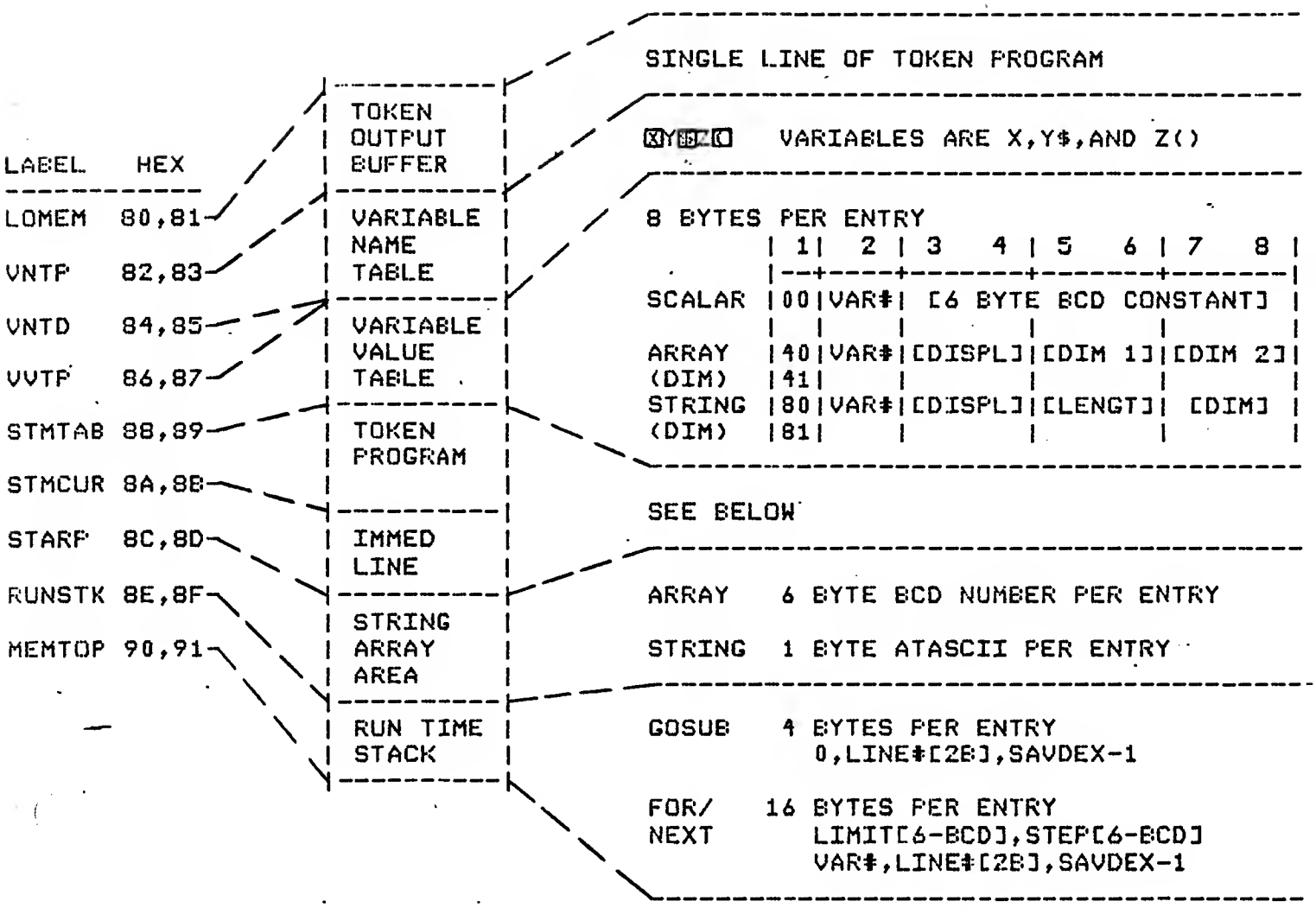
- A. Speeding Up BASIC
- B. Saving Memory
- C. BASIC Bugs

II. HOW ATARI BASIC WORKS

- A. The Token File Structure
  1. Token Output Buffer
  2. Variable Name Table
  3. Variable Value Table
  4. Tokenized Program
  5. Immediate Mode Line
  6. String Array Area
  7. Run Time Stack
- B. LOAD and SAVE BASIC Files
  1. Format Of SAVEd Disk File
  2. How The Pointers Are Used

III. ADVANCED PROGRAMMING TECHNIQUES

- A. Special Features Of BASIC And The OS
- B. Example Programs
  1. Initialize A String
  2. Delete Lines Of Code
  3. Modify String/Array Pointers
  4. Save And Retrieve BCD Numbers On Disk
  5. Name Table Modifier



EXAMPLE PROGRAM

```
10 REM TOKENS
20 FOR X=PEEK(130)+PEEK(131)*256 TO PEEK(140)+PEEK(141)*256-1
30 PRINT PEEK(X);" ";:NEXT X
```

EXAMPLE OUTPUT (PARTIALLY FORMATTED FOR READING)

```
RUN

(VNT) 216 ([X])
(VND)   0 (DUMMY)
(VVT)   0   0  65 118  51   0   0   0
(STM)  10   0  12  12   0  84  79  75  69  78  83 155 (ATASCII)
       20   0  75  75   8 128  45  70  58  14  65   1  48   0   0   0  44  37
              70  58  14  65   1  49   0   0   0  44  36  14  65   2  86   0
               0   0  25  70  58  14  65   1  64   0   0   0  44  37  70  58
              14  65   1  65   0   0   0  44  36  14  65   2  86   0   0   0
              38  14  64   1   0   0   0   0  22
       30   0  19  15  32  70  58 128  44  21  15   1  32  21  20  19   9 128
              22
(IMM)   0 128   6   6  37  22
```

NOTE - BOXED CHARACTERS APPEAR AS INVERSE VIDEO ON THE SCREEN

BASIC BUG LIST

1) An INPUT statement with no variable is not flagged as an error.

2) PRINT A=NOT B crashes the system.

3) DIM L (10) is treated as DIM L10).

4) LOG(0),CLOG(0),LOG(1),CLOG(1) give incorrect values.

5) PRINT -0 gives a meaningless number.

6) Line editing lockup and incorrect handling of 256 byte strings.

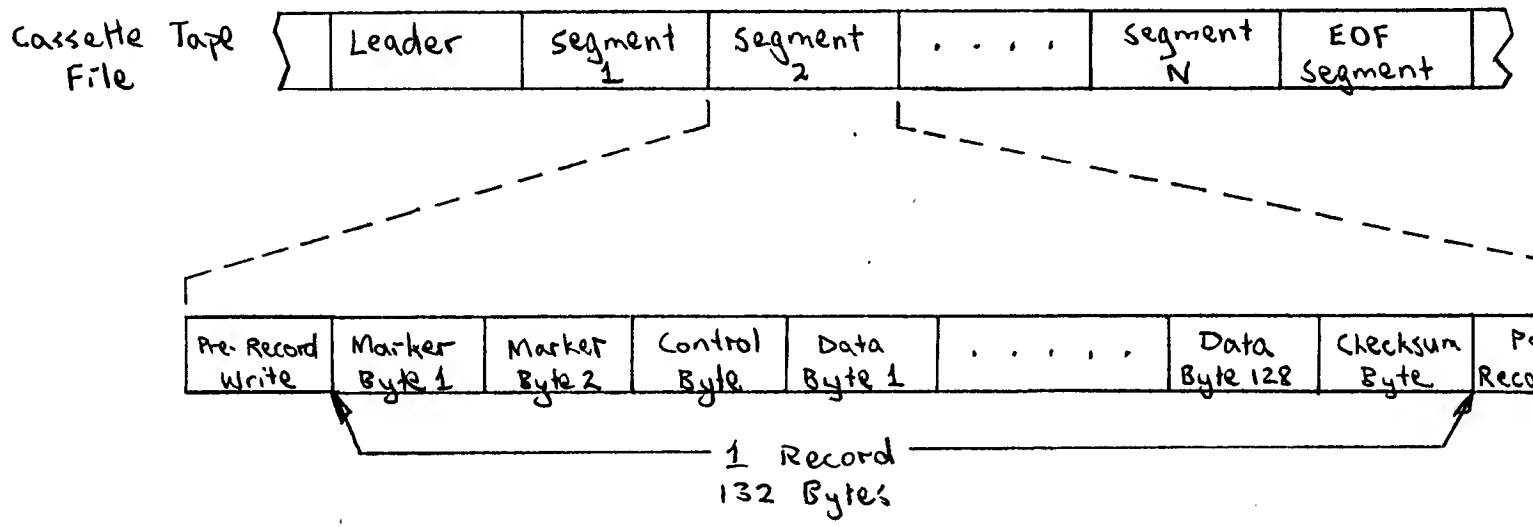
7) A numeric array of 32766 by 32766 elements can be dimensioned.

8) A RESTORE command does not generate an error 12 if the restored line number does not exist.

9) Integer exponents of integer values do not give correct results.

10) An input from disk of 256 bytes will overwrite the first 128 bytes of page six in RAM.

11) A PRINTed CTL R or CTL U is treated as a semicolon.



CASSETTE TAPE INFORMATION

Mark Tone (1) = 5327 Hz
Space Tone (Ø) = 3995 Hz

Leader : 20 Sec Mark Tone

Pre-Record Write (PWRT):
Mark Tone

Post Record Gap (PWG):
Short: Mark, Normal: Unknown

Inter-Record Gap Times

	PWRT	PRG
Short	0.25 sec	0-N sec
Normal	3 sec	≤1 sec

Marker Bytes = $55

Control Byte Options

$FC Full data record (128 bytes)

$FA Partially full - count is in byte prior to checksum (data byte 128)

$FE End Of File record with 128 zero bytes

Checksum Algorithm

→ Partial Sum
+ Current Byte
+ Carry
Result

OPERATING SYSTEM OUTLINE

1) System Routines (No direct user access)

   Monitor
      Handles Power-Up and RESET Button
      Boots in application software
      Has its own "Memo Pad"

2) System Tables (ROM and RAM)

   System Utility Vectors
      Starting addresses of various OS utilities

   System Database
      Pointers and variables used by the system

   Hardware Shadow Registers
      Allows updates to occur during VBLANK

   Character Set
      The 128 ATASCII characters

   Various ROM Tables
      Keyboard to ATASCII translation
      Screen handler tables
      Additional tables

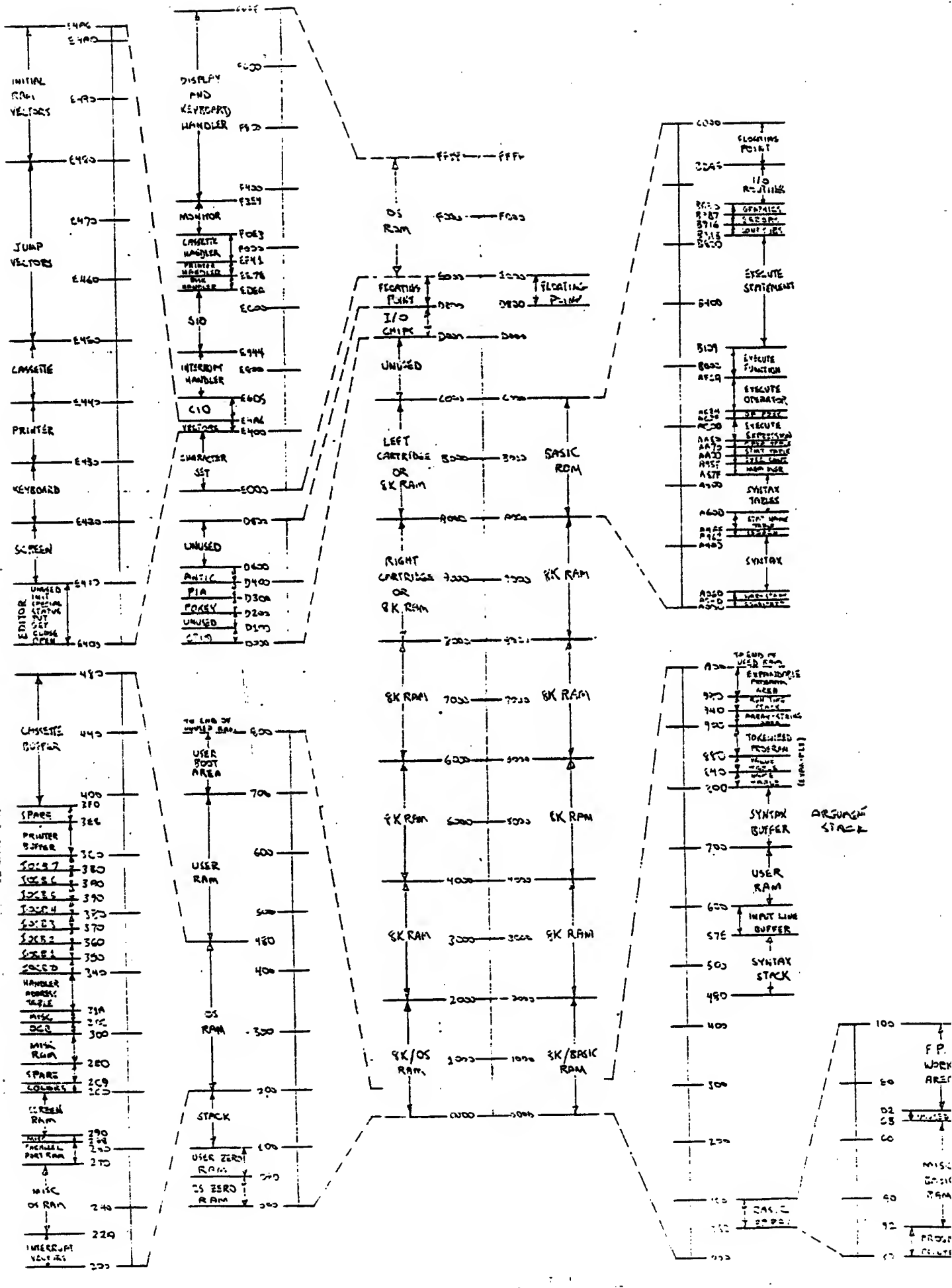
3) User Callable System Utilities (Direct user access)

   I/O Subsystem
      Central I/O Utility (CIO)
      Device Handlers
      Serial I/O Handler (SIO)

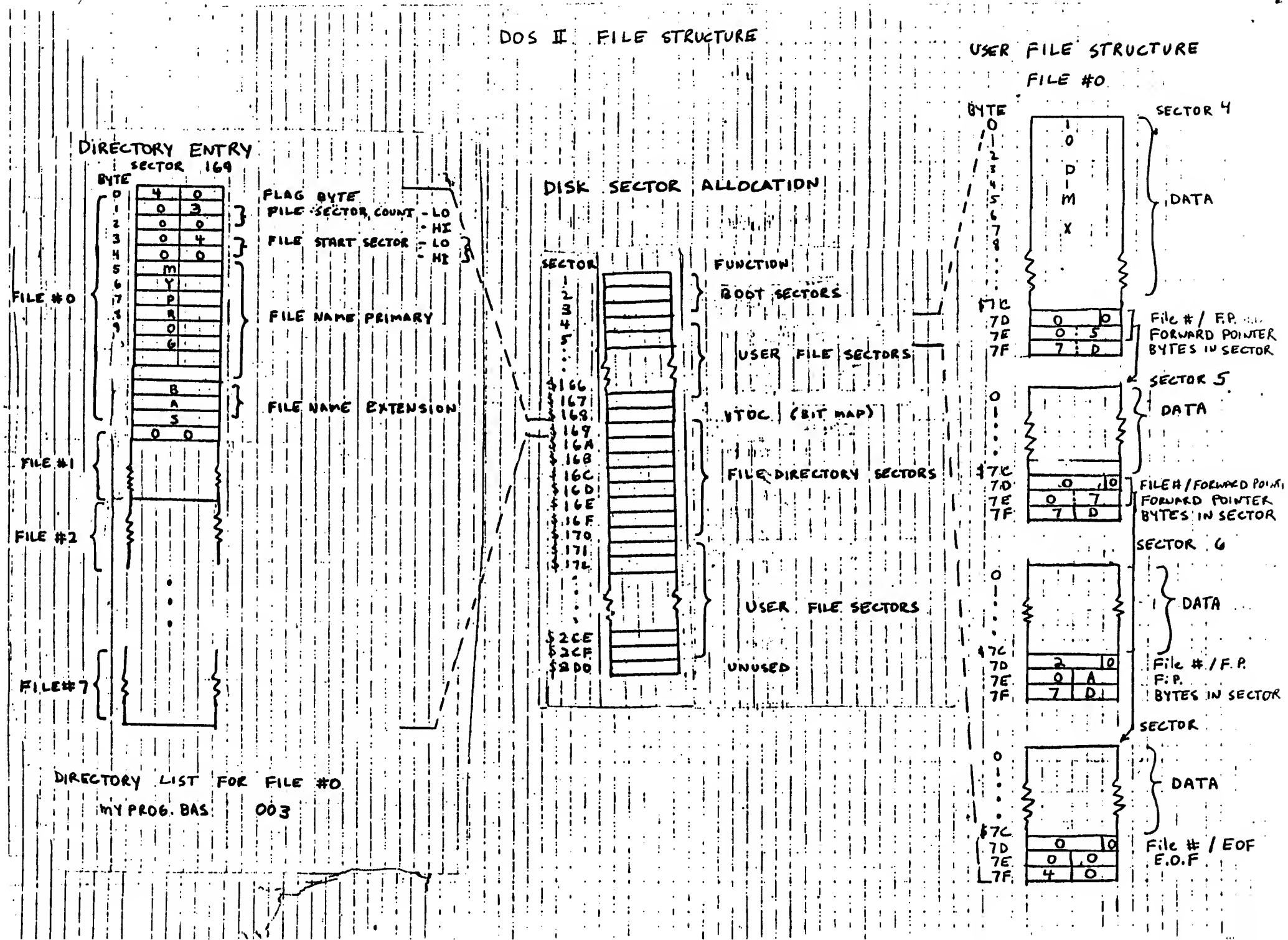
   Interrupt Handlers
      NMI
      IRQ

   Floating Point Routines
      Six byte BCD format used

   Software Timer Routines
      Allow timing of 1/60 second resolution


INITIAL RAM VECTORS
JUMP VECTORS
CASSETTE
PRINTER
KEYBOARD
SCREEN
EDITOR
DISPLAY AND KEYBOARD HANDLER
MONITOR
CASSETTE HANDLER
PRINTER HANDLER
RS232 HANDLER
SIO
INTERRUPT HANDLER
CIO
VECTORS
CHARACTER SET
UNUSED
ANTIC
PIA
POKEY
UNUSED
OS ROM
FLOATING POINT
I/O CHIPS
UNUSED
LEFT CARTRIDGE OR 8K RAM
RIGHT CARTRIDGE OR 8K RAM
BASIC ROM
8K RAM
8K RAM
8K RAM
8K RAM
8K RAM
8K/OS RAM
8K/BASIC RAM
FLOATING POINT
I/O ROUTINES
GRAPHICS
SECONDARY
OUTPUT
EXECUTE STATEMENT
EXECUTE FUNCTION
EXECUTE OPERATOR
SYNTAX TABLES
SYNTAX
CASSETTE BUFFER
SPARE
PRINTER BUFFER
MISC RAM
SPARE
SCREEN RAM
MISC OS RAM
INTERRUPT VECTORS
USER BOOT AREA
USER RAM
OS RAM
STACK
USER ZERO RAM
OS ZERO RAM
EXPANDABLE PROGRAM AREA
TOKENIZED PROGRAM
SYNTAX BUFFER
USER RAM
INPUT LINE BUFFER
SYNTAX STACK
ARGUMENT STACK
BASIC
FP WORK AREA
MISC BASIC RAM
PROG

DOS II FILE STRUCTURE



ATARI UTILITIES DISKETTE

This utilities diskette is distributed to outside software developers to facilitate their efforts. This software is not in the public domain and is not to be distributed. The utilities in this package are:

DOS	SYS	039	DOS 2.0S
DUP	SYS	042	DOS 2.0S
BUILD24		010	builds self-booting BASIC programs
DIV	SRC	026	an integer divide utility
BMUL	SRC	020	a signed integer multiply routine
SMUL	SRC	027	unsigned integer multiply routine
CHRGEN		049	a simple character set editor
SOUND		095	a sound editor
XREF		052	cross-reference program for BASIC
MASHER		045	compresses BASIC programs
RENUM	BAS	030	renumbers BASIC programs
FORMTR		004	formats LIST files from ASSEMBLER
M8TXT		005	BASIC mode 8 character print routine
BCDSAV		004	makes fixed-length number records
HOBBY1		008	simple player-missile graphics demo
HOBBY2		005	simple display list interrupt demo
SCRL19	ASM	104	full fine scrolling module
SCRL19	OBJ	011	object code
FIX	OBJ	042	a disk sector utility
SCRLH	DEM	004	simple horizontal scroll demo
MDIR	OBJ	002	gives disk directories from BASIC
HORSE20	BAS	047	a character set animation demo
AUTORUN	SYS	001	the R: device handler
MDLI02	BAS	005	multiple display list interrupt demo
DEMO3	BOK	005	player priority trick
DEMO4	BOK	004	player as special character demo
DLISTA	DEM	005	alternating display list demo
SCRLF	DEM	003	simple fine scroll demo
SCRLV	DEM	004	simple vertical scroll demo

None of these programs are finished products; they all contain rough spots. If you make improvements please send a copy to our library.

CHARACTER EDITOR

This program is a character editor that makes it easier to take advantage of the redefinable character set capability of the Atari. It gives you the capability to edit, load, or save character sets.

The first menu option is to create or edit a character set. If you enter this option without first loading a character set, it will default to a blank character set. You edit characters with the joystick, selecting a pixel position with the stick and the status of that pixel (on or off) with the button. When you are done editing a character, you can allocate it to a character position with a keystroke indication. Fear not, most operations are prompted, so that you can pick your way through the program rather well. The only blooper is that the prompt for an I/O operation to the disk requires that you give the D1: prefix.

This program is usable but not at all as practical as IRIDIS's FONTEDIT program. I strongly urge you to buy and study the IRIDIS program if you want to do any character set work. If only it had a display list interrupt....

SOUND EDITOR

The sound editor helps you develop new sounds. It is not appropriate for developing tunes or jingles, or any long sound. It is designed for developing short sounds (1 second long) such as clangs, croaks, rattles, and other such nonsense. It only edits two of the four sound channels.

The program needs very little external documentation, as its title page describes the commands. The joystick response is slow, but you can use the 'fast' command to speed it up and then use the normal speed to fine tune your sound. You should read the hardware manual to get an idea of what the sound registers do.

Don't overlook the possibilities this program opens up. I have heard some very convincing sound effects created with it; it just takes a little imagination.

CROSS REFERENCE UTILITY

This utility provides a cross-reference of variables and constants in a BASIC program. It requires at least 40K of RAM, a printer, and a disk. The BASIC program to be cross-referenced should be on a diskette in SAVE format. The program first gives a count of the total number of variables used. For each variable, it lists all line numbers containing references to the variable. It also gives a count of how many times each constant is used. If an error occurs during printer output, you may recover by typing GOTO 3050.

RENUMBER

This program will renumber your BASIC program on disk. The target program must be on the diskette in LIST format. The program prompts you for the values it needs. The 'input device' will normally be D:progname. The 'output device' will normally be D:newname. The 'starting number' is the new starting line number. 'From' and 'to' are the beginning and ending line numbers of the section of code you want renumbered.

FORMATTER

This program will format the output of your Assembler/Editor cartridge so that you can get a list file that looks good out of your Atari 825 printer. Make the first instruction of your assembly code a .OPT NOEJECT. Then assemble the list file to the diskette. Then drop into BASIC and run this program. Respond to the name prompt with D:progname. This program is designed for use with the Atari 825 printer, so good luck with anything else.

MAPSCROLL

Mapscroll is a demonstration program module that shows one way to use fine scrolling effectively. It creates a large map in BASIC's graphics 2 mode using a redefined character set. The map is 32x64 pixels in size, but only 10x20 pixels are displayed on the screen at any one time. The user can scroll the screen window across the map with the joystick. The program was written for easy integration into other packages.

Scrolling is achieved by coupling fine scrolling through the hardware fine scrolling registers with character scrolling by modifying the LMS bytes in the display list. The fine scrolling is straightforward; the character scrolling is somewhat more intricate. Each display byte in the display list has its LMS bit set. The following two bytes give the address of the display data. When the fine scrolling register overflows in the scrolling routine, the routine adjusts the bytes in the LMS addresses to point to the next character bytes. This adjustment is kept track of through a variable referred to as the offset.

The other trick in the program is the redefinition of the character set into a graphics character set. The technique is very powerful; very few of the available characters in the character set are used and yet the resulting map is very believable. The map could be made even more realistic by using the other characters. By changing the character set at appropriate times the program could produce a variety of effects.

The amount of system resource used is low. The module as written occupies 4K of RAM. This includes the map, the display list, the initialization routines, and the interrupt service routine that reads the joystick. Outside of this the program uses 4 bytes of page zero (two of which are available after initialization is complete) and seven bytes of page six. The interrupt service routine is very fast so it will not significantly slow whatever main program you plug it into. Space has been left inside the 4K block for additions and modifications. The program is not fully relocatable as there are four patches that must be made to relocate it; however, these patches are well documented and easy to do.

The easiest way to see this program in action is to BINARY LOAD it from DOS. The file name is SCRL19.OBJ. Then call it from BASIC with A=USR(27648). Plug a joystick into port 1 and scroll. At present the program does not interface well with BASIC; I have found that BASIC's cursor positioning gets lost. This reflects more on my laziness than on the program's tractability.

D. Yocum
4/9/80

CARE AND FEEDING OF "MASHER"

MASHER is a utility program writen in BASIC which compreses other BASIC programs. Since it is intended for internal use only, there are a few "features" which the user should be aware of.

1) All files are in LIST format.

2) Do not use lines 0-9 in the source program. MASHER will use these lines for it's own variable definitions.

3) Do not use the variables Q0 - Q999. MASHER uses these for constants.

4) Do not branch to REMark statements. Besides being bad pratice, MASHER has problems sometimes with this.

5) There is presently a bug with DATA statements. They are packed like any other statement. You will need to unpack these after MASHing. This bug will be fixed (someday).

WHAT IT DOES

MASHER will perform the following conversions on the source program

1) Removes all REMarks

2) Converts frequently used constants to variables.

3) Packs small lines together to form long lines.

NOTES

This program will usually save between 5% - 40%. Maximum compression occures when short lines are used in the source program. You must know the number of variables used in the source program. This can be obtained by running XREF ~~SYMBOL~~.

PMMOVE

This program uses a machine language routine to move Players and Missiles. The routine is called by:

```
USR(ADR(MOVE$),PMNUMB,XPOS,YPOS)
```

PMNUMB refers to the Player (0-3) or Missile (4-7) to move. The Missile number is determined by PMNUMB-4. XPOS and YPOS are the X and Y coordinates to place PM. A Relocator is used to make the PM Move routine wherever it may be placed in RAM by BASIC.

BINARY ROUTINE

This routine is included as part of the PMMOVE routine. BINARY loads or saves a binary file from BASIC.

```
On entry  CMD=7    means load a file
          CMD=11   means save a file
          STADR=   the address to load or save a file from
          BYTES=   the number of bytes to save or load
          IOCB=    the IOCB to use
          FILE$=   file name to load

On exit   ERROR=1  means successful load
          ERROR<>1 means it's an error status
```

BMUL

This routine implements BOOTHS ALGORITHM for multiplication of SIGNED binary numbers in TWO's-COMPLEMENT notation.

The MULTIPLIER is shifted to the right with the PRODUCT (as usual). Each CHANGE of the MULTIPLIER bits from zero to one causes the MULTIPLICAND to be subtracted from the PRODUCT. Each change from one to zero causes it to be added.

Like most signed arithmetic, it cannot be chained and is prone to overflow problems when given -32768, but it is smaller than the absolute-kludge wrapped around an unsigned mutiply, and not much slower (633-945 cycles).

```
Enter with      A,Y = MULTIPLIER (A=MSB)
                ACC = MULTIPLICAND

                 16 * 16  SIGNED MULTIPLY

Exit with    ACC,MQ = 32 BIT PRODUCT
                ACC = MSW
                 MQ = LSW

      ACC = 212
       MQ = $0CB
      ENT = $0CD
       SC = $0CF
        * = $600
```

DIV

This routine is composed of two assembly subroutines: Unsigned Divide of 32/16 bits and Signed Divide of 16/16 bits.

Unsigned Divide:

```
Enter with A,Y = Divisor (A=MSB)
        ACC,MQ = Dividend

Exit with  ACC = Remainder
            MQ = Quotient
```

Signed Divide:

```
Enter with A,Y = Divisor (A=MSB)
           ACC = Dividend

Exit with  ACC = Remainder
            MQ = Quotient
```

SMUL

This routine is composed of a 16 * 16 Unsigned Multiply and a 16 * 16 Signed Multiply.

Unsigned Multiply:

```
Enter with A,Y = MULTIPLIER
           ENT = MULTIPLICAND
           ACC = ADDEND
      (ACC,MQ) = ([A,Y]*ENT)+ACC
      (This way for chained operaton)
       556-748 Cycles
```

Signed Multiply:

```
Enter with A,Y = MULTIPLIER (A=MSB)
           ACC = MULTIPLICAND

Exit with  ACC,MQ = PRODUCT
              ACC = MSW

584-821 Cycles
```

M8TXT

M8TXT demonstrates how to mix text with graphics in BASIC mode 8. It plots the characters bit by bit onto the mode 8 screen. The technique can be adapted to any program using BASIC mode 8 displays.

BCDSAV

This program provides an alternative to the variable length records obtained when a program PRINTs values to the disk. Using the variable table values, it stores BCD values of numbers to the disk in fixed length records.

HOBBY1

This is a simple player-missile graphics demo. It sets up a player and then moves it around with the joystick. Since this is a pure BASIC program, the vertical motion of the player is too slow. An assembly language routine is necessary to get proper high-speed motion.

HOBBY2

This is a simple display list interrupt demo program. The bottom half of the screen changes from blue to pink.

BUILD24

This program creates an AUTORUN.SYS file that will start up your BASIC programs. It asks you for a BASIC command; the command you enter will be in the AUTORUN.SYS file and will be executed on powerup. Normally your command will be of the form RUN"D:PROG.BAS". It must be less than 128 characters long. Remember to put in the terminating double quotation mark (").

MDIR

This program is an object file that can be called from BASIC to put the disk directory onto the screen. To use it, you must first load it into RAM with the BINARY LOAD command (L) in the DOS. Call it from BASIC with A=USR(1536).

AUTORUN.SYS

This program is an RS-232 handler file. It allows you to use the R: device. It is booted in automatically on powerup.

HORSE20.BAS

This is the running horse demo. It demonstrates the power of character set graphics and animation.

SCRLH.DEM

This is a simple horizontal coarse scroll demo.

SCRLV.DEM

This is a simple vertical coarse scroll demo.

SCRLF.DEM

This is a simple fine scroll demo.

MDLI02.BAS

This is a multiple display list interrupt demo. It puts gobs of color onto the screen. Do not be alarmed if the screen goes black; it takes several minutes to finish. Once it is running observe how keypresses affect the display. Also note the greatly reduced computation speed of BASIC. With so many interrupts happening, the 6502 has little time for other activities.

DEM03.BOK

This program demonstrates a technique for increasing the resolution of a stationary player by hiding it behind a playfield cutout.

DEM04.BAS

This program shows how a player can be used as an extended character.

DLISTA.DEM

This program demonstrates the alternating display list technique.